1. System Message Services (SMS)

1.1 System Message Services Introduction

All previous references to HCI (Human Control Interface) will now be know as the CCWS(Command Control Work Station). Shaded diagrams are relative to SMS.

1.1.1 System Message Services Overview

System Message Services is an integrated support service which provides applications the ability to send and receive system and application message packets across the network. System Message Services executes on the Gateways, CCP, DDP, CCWS(HCI), and Ops CM Server platforms. System Message Services receives messaging information from the calling application and forwards this message packet information to CCWS workstations for translation and display, in addition to logging packets to the SDC Recording Facility and the SM Master log file. System Message Services minimizes network traffic by utilizing a Central Message Repository(Online Message Catalog) that contains the encapsulated CLCS Message Catalog eliminating transmittal of the message body (i.e., actual text message). The catalog is referenced to obtain information that is associated with each message at the destination platform.

To send messages, applications call the SMS SMS_Send_System_Message API and supply a TCID or SCID parameters that uniquely identifies the index file to access in order to describe the message characteristics.

System Message Services also provides the capability to retrieve system and application messages from the SDC Recording Facility through a programmatic interface(API). Figure 1 is a conceptual view of the path taken by the Message calls.

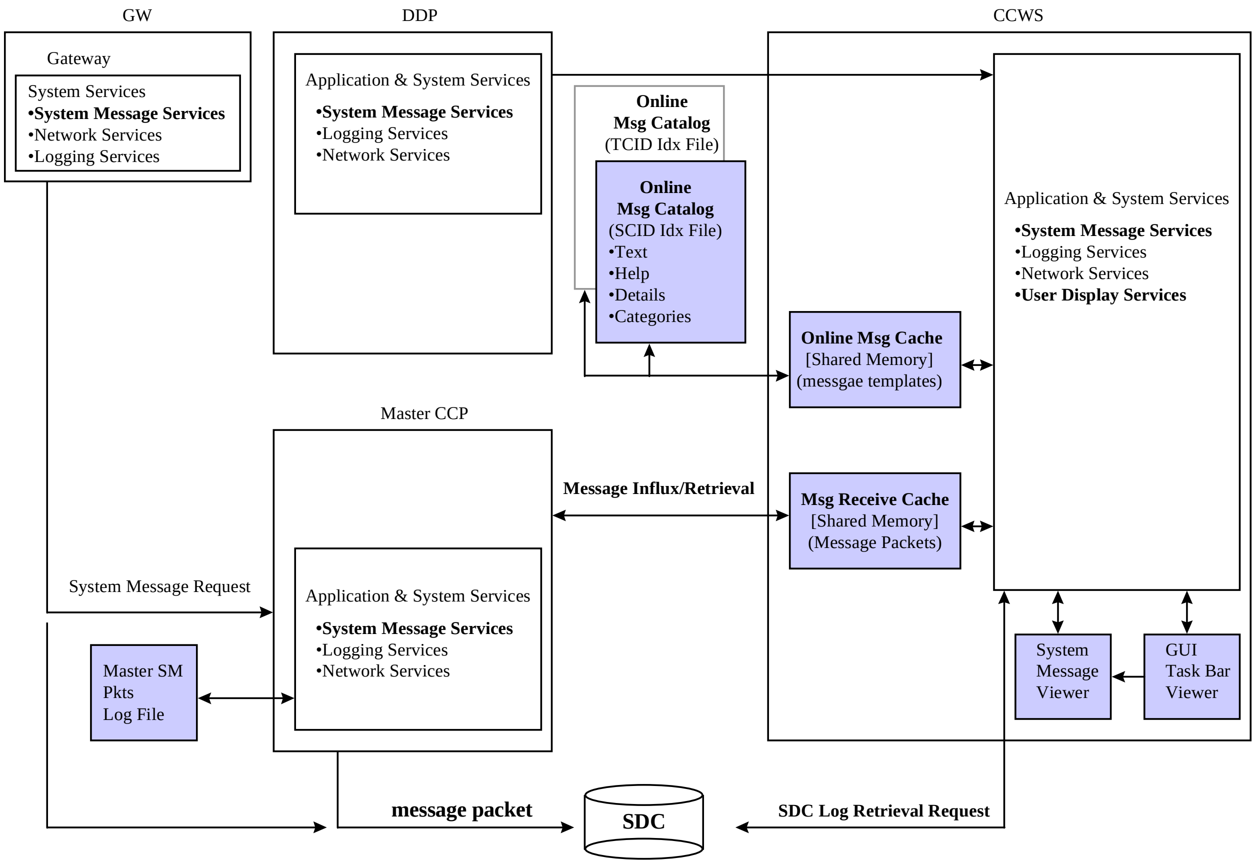


Figure 1. (Message Packet Call Path)

1.1.2 System Message Services Operational Description

Applications send messages via the SMS Send_System_Message API call. Those applications executing in the Gateway issue the SMS Send_System_Message API which builds the message packet that is forwarded to the CCP's Message Router Process. The CCP's Message Router Process utilizes CLM/RM Network Services to read those message packets and then forwards those packets to all CCWS(s). Additionally, the SMS Message Router Process also records those message packet received to the SDC and the SM Master Log File. The following figure (figure 2) delineates the flow of message packets from all platforms. Any non-gateway path is that taken by message packets sent from platform other than the gateway origin.

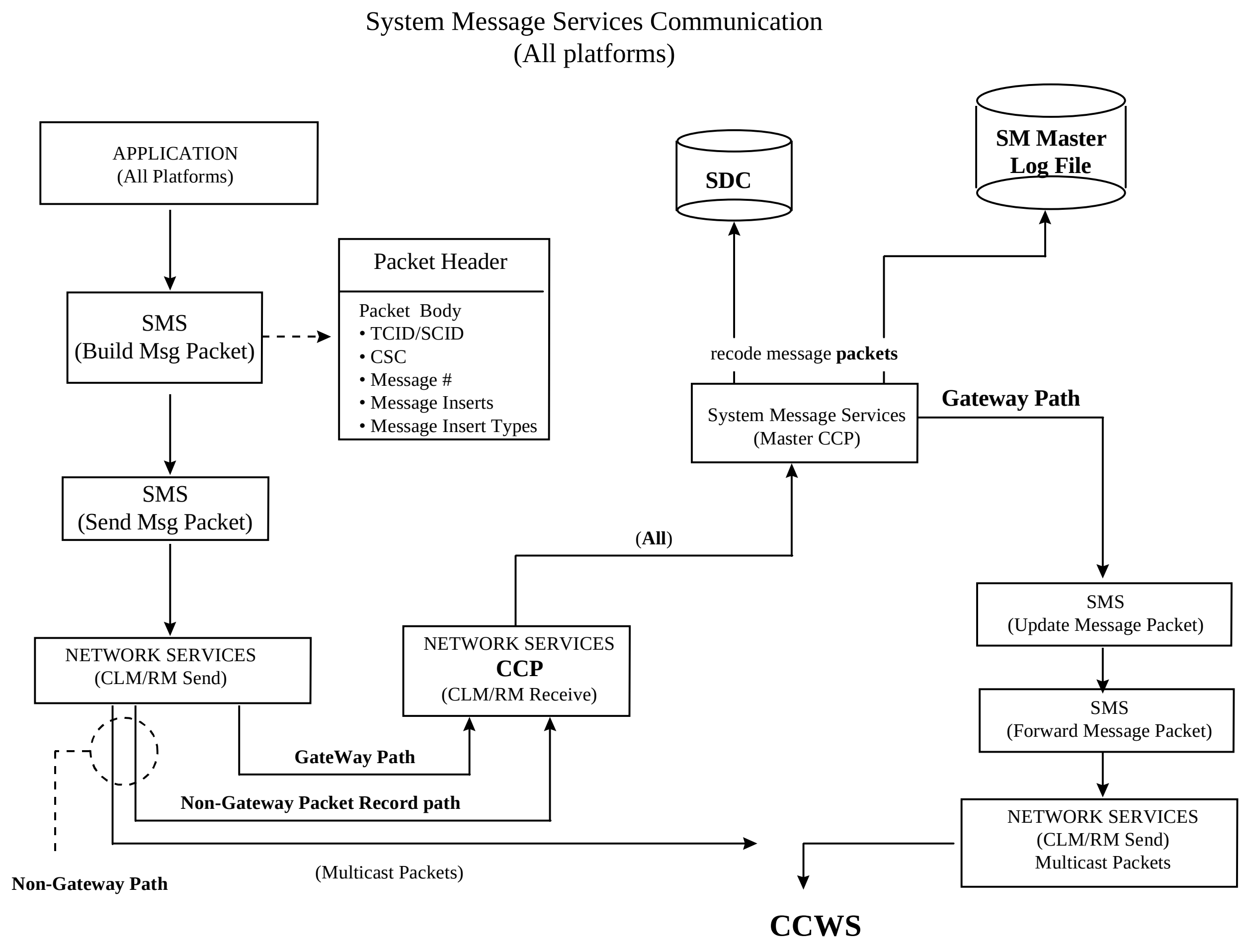

System Message Services Communication
(All platforms)
APPLICATION (All Platforms)
SMS (Build Msg Packet)
Packet Header
Packet Body
• TCID/SCID
• CSC
• Message #
• Message Inserts
• Message Insert Types
SMS (Send Msg Packet)
NETWORK SERVICES (CLM/RM Send)
NETWORK SERVICES CCP (CLM/RM Receive)
GateWay Path
Non-Gateway Packet Record path
Non-Gateway Path
(Multicast Packets)
(All)
SDC
SM Master Log File
recode message packets
System Message Services (Master CCP)
Gateway Path
SMS (Update Message Packet)
SMS (Forward Message Packet)
NETWORK SERVICES (CLM/RM Send) Multicast Packets
CCWS

Figure 2. (SMS Communications—all platforms)

Applications executing on the CCP, DDP, and CCWS issue the same SMS_Send_System_Messgae API. SMS builds the message packet that is sent to all CCWS's and records each packet to the SDC and the SM Master Log File. Each CCWS has a SMS Message Router Process that utilizes CLM/RM Network Services to read incoming message packets while populate the SMS Message Receive Cache (shared memory segment) in order for multiple viewers to receive messages simultaneously. Additionally, each viewer maintains a dedicated read thread that retrieves those message packets sent to the CCWS shared memory area and delivers them to the SMV viewer.

Reference Figure 3. SMS Communications (DDP/CCP/CCWS/Ops Server platform)

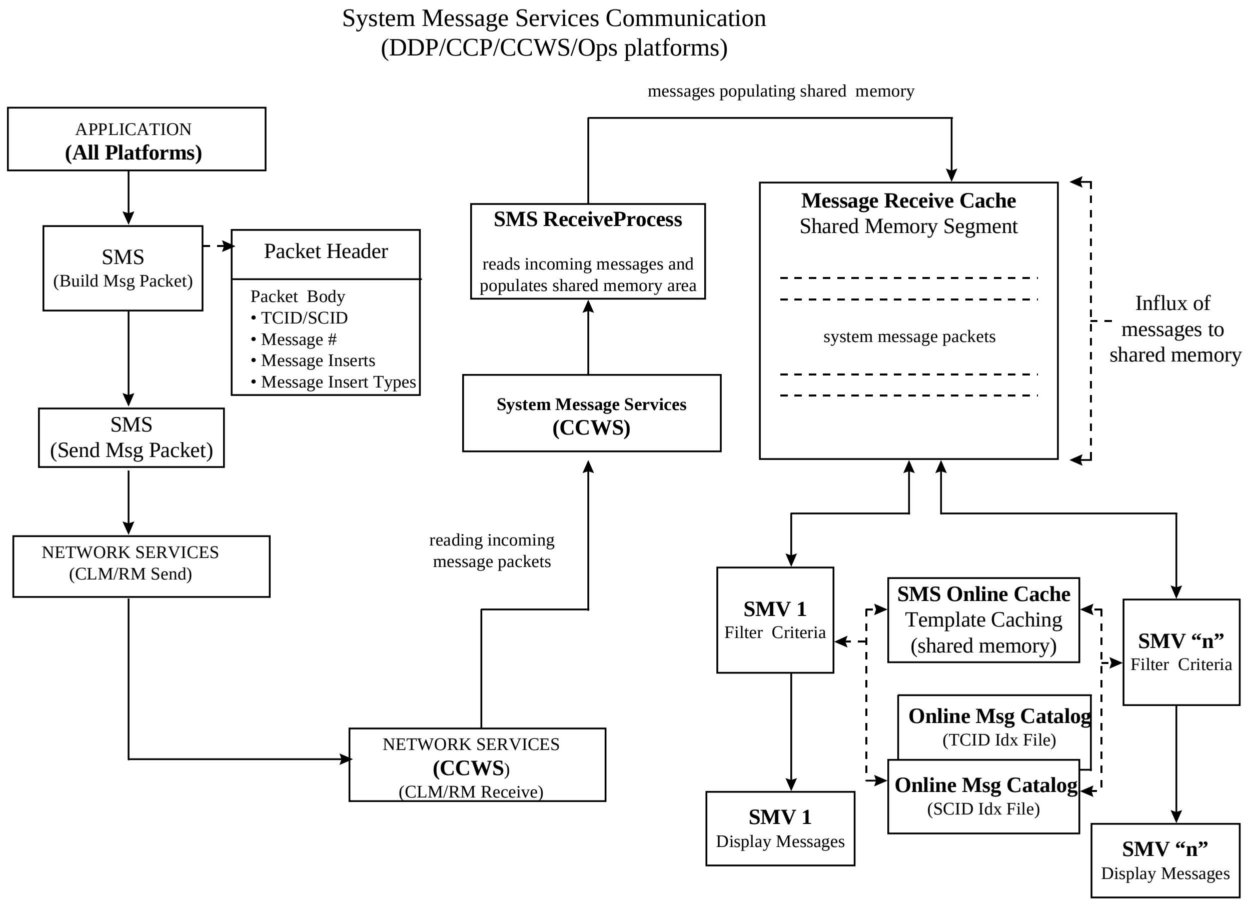


Figure 3. SMS Communications (DDP/CCP/CCWS/Ops Server platform)

1.2 System Message Services Specifications

1.2.1 System Message Services Ground rules

- The Message Catalog and Help Information will consist of an indexed files that will be supplied to the CM System Build and Test Build for download.
- Test Load must load the TCID Message Index File on all CCWS platforms.
- System Load must load the SCID Message Index file on all CCWS platforms.
- The Message Text and help will not be transmitted with every message across the system.
- The System Message Viewer (SMV) must display an appropriate message whenever SMS cannot successfully retrieve or format the message text from the Message Index file.
- Flow control of system messages is the responsibility of the application.
- Application Services will provide a mechanism for retrieving the subsystem name.
- Startup of the SMS Receive Process is launched by Subsystem Services during CCWS boot time.
- Message packets will populate the Message Receive Cache Contiguously. This will prevent any fragmentation of space in the shared memory segment and allow for a greater number of packets in this area.

1.2.2 System Message Services Functional Requirements

The System Message Services (SMS) provides a method for applications to send and receive messages.
Requirements that are post-Thor are in italics.

1. SMS will provide an API to send messages from any workstation application to:
    a) All workstations (within the same activity)

2. SMS will distribute the message packet to multiple RTPS destinations and to the SDC for recording.
3. SMS will format the message to be sent to the SDC for recording.

4. Each Message Packet will contain the following minimum contents:
    a) *Source - Application ID*
    b) subsystem name
    c) *RefDes*
    d) Message Number
    e) Message Parameters
    f) Time Stamp

5. Each message from the indexed message file will contain the minimum contents:
    a) CSCI
    b) CSC
    c) Severity Level
    d) RSYS
    e) Message Type

6. SMS will time stamp each message using JTOY format.
7. SMS will provide a mechanism to allow an audible alarm to be activated in conjunction with the message.
8. SMS API will return success or failure status back to calling application.
9. SMS API will provide the capability for specifying message parameters(positional and non-positional).
10. SMS API allows a maximum of 30 message parameters (i.e., message inserts) to be specified.
11. SMS API will allow the user the capability to send up to a maximum of **512** bytes of message data.
12. SMS API will perform validation of each message's parameter passed from an application.
13. SMS will provide the capability to receive and process messages sent from applications residing on either a local or remote workstation.

14. SMS will utilize Application Services API's to format UTC/CDT time.
15. SMS will provide an API for retrieving system message packets from the SM Master Log File.
16. SMS will provide the capability to include the origin (CPU_ID) translation) of the system message.
17. SMS API will provide the capability to retrieve insert details from the System Message Index file (TCID/SCID File).
18. SMS API will provide the capability to retrieve Message Help Text associated with the system message (TCID/SCID).
19. *SMS API will provide the capability for retrieving messages packets from the SDC, via an SDC API call when requested by the operator.*
20. *SMS API will provide the capability to retrieve message packets based on the start time of the messages .*
21. *SMS will retrieve the message packets from the SDC if the number of packets requested is not available on the SM Master log file.*

1.2.3 SMS Performance Requirements

None identified for Thor.

1.2.4 SMS Interfaces Data Flow Diagrams

This section provides a description and diagram of all of the interfaces to SMS.

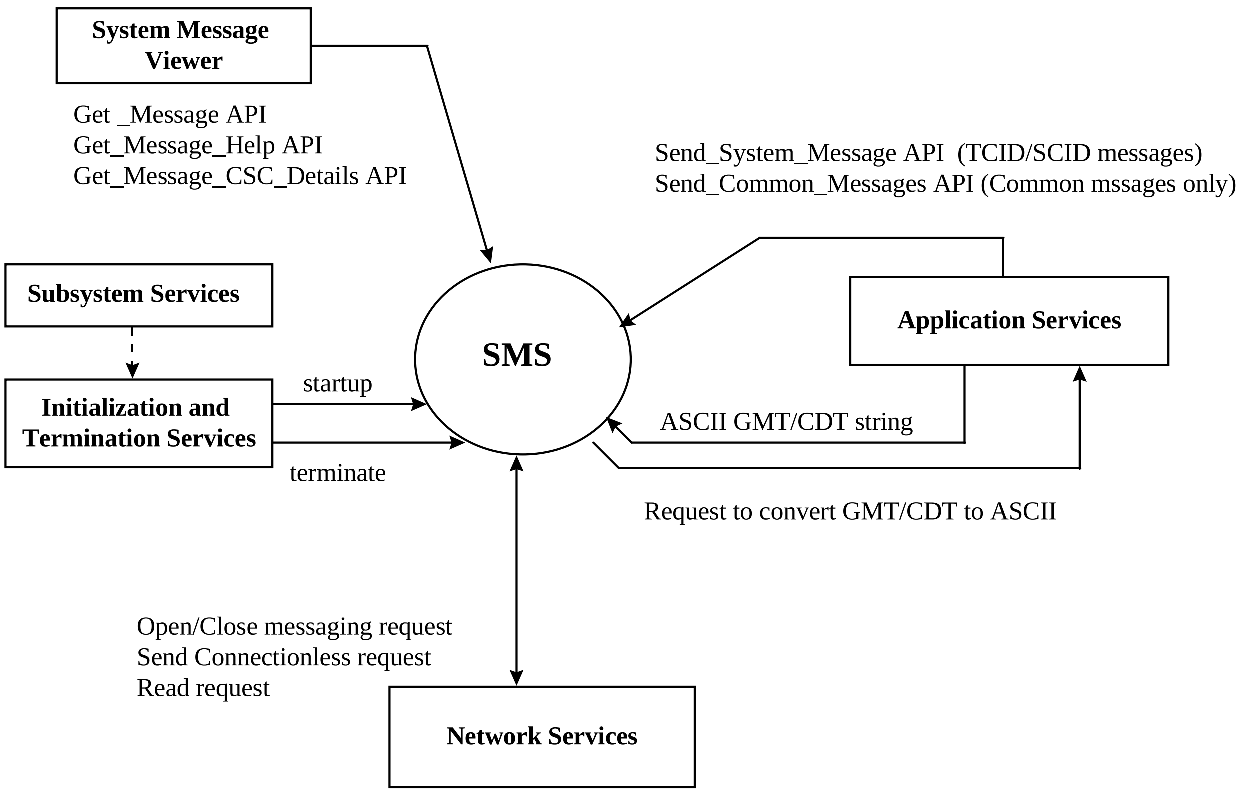

System Message Viewer
Get _Message API
Get_Message_Help API
Get_Message_CSC_Details API
Send_System_Message API (TCID/SCID messages)
Send_Common_Messages API (Common mssages only)
Subsystem Services
SMS
Application Services
Initialization and Termination Services
startup
terminate
ASCII GMT/CDT string
Request to convert GMT/CDT to ASCII
Open/Close messaging request
Send Connectionless request
Read request
Network Services

Figure 4. SMS Interface flow

System Message Services contains interfaces with Applications, System Message Viewer, Initialization and Termination Services, Network Services, Local Logging Services, Conversion routines.

Subsystem Services will starts the SMS Router and SMS Receive Process following platform startup.

SMS interfaces with Network Services (on the CCP platform and CCWS platforms) to register and receive messages on the network.

Applications send messages via the SMS_Send_System_Message API with the parameter inserts included in the API call. Applications also send Common and TCID messages by issuing the Send_Common_Message and Send_TCID_Message API's. These additional API's are being introduced to reduce the modification impact on the application community. In future releases all message sending API's will be reduced to a single call.

The System Message Viewer issues the SMS_Get_Message API to read the text message from the Message Index file, and uses the SMS_Get_Message_CSC_Details to obtain details on the message. The System Message Viewer issues the SMS_Get_Message_Help API to read the help text.

SMS, when formatting the inserts to the message text, calls the Conversion Routines to convert GMT and CDT to ASCII GMT and CDT, respectively.

1.3 SMS Design Specification

SMS is comprised of two daemon processes, SMS Router and SMS Receiver, and a number of API's used by applications (including the System Message Viewer) to use the services. The SMS Router and SMS Receiver process calls Network Services to handle the sending and receipt of messages. The SMS Router Process executes on the Master CCPs to handle message packets from the gateways. The SMS Receiver Process executes on all CCWS platforms to handle all incoming message packets from all other platforms (i.e., DDP, CCP, CCWS), and is launched by Subsystem Services when the CCWS is started. When an application issues the SMS_Send_System_Message API, SMS utilizes CLM when sending a message payload packet on the network.

1.3.1 SMS Detailed Data Flow

The following data flow provides a pictorial representation of the data flow between external sources and destinations, and the major and minor functions of SMS. The message is received on all CCWSs.

Detailed Data Flow Diagram

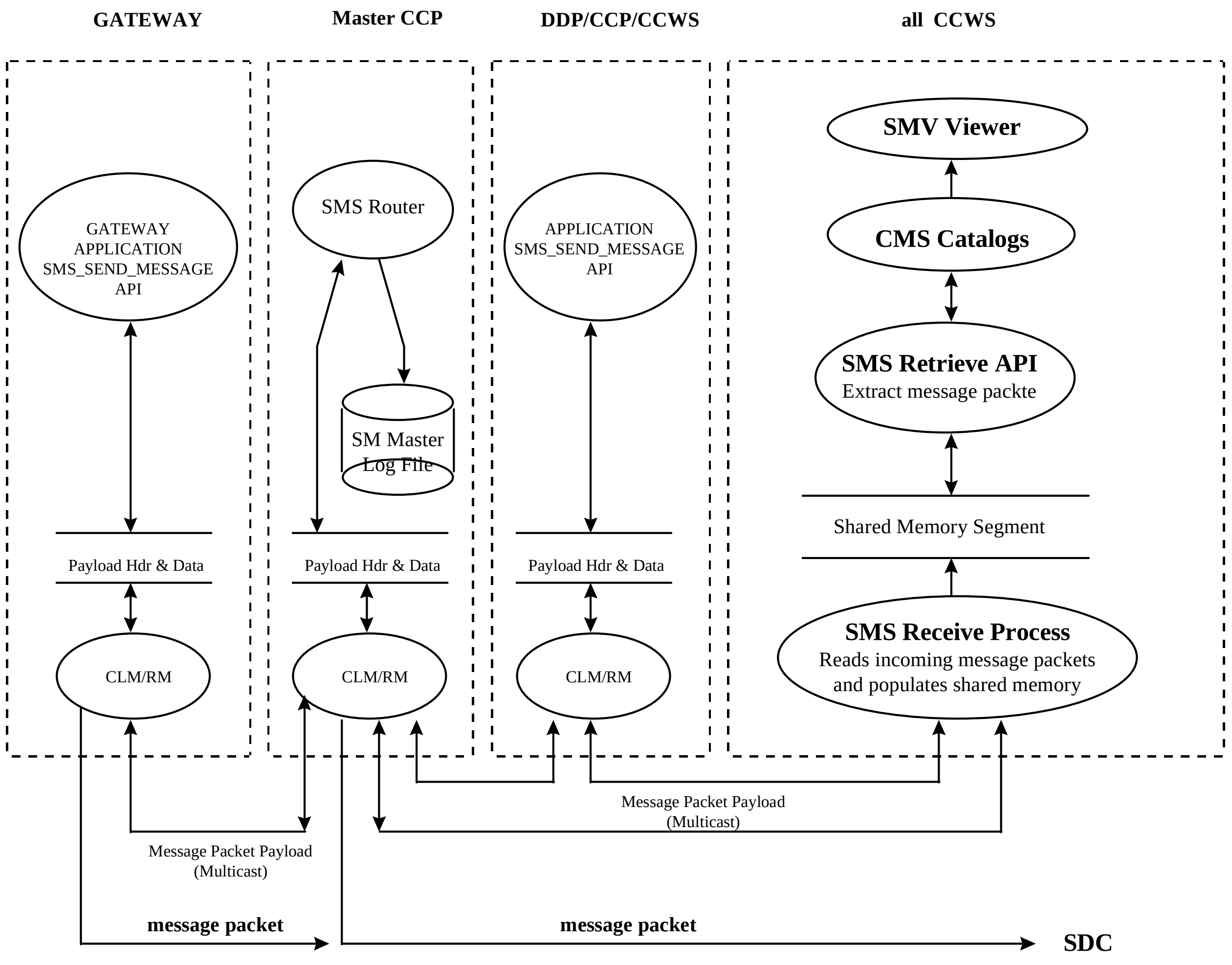


Figure 5. Detailed data Flow

1.3.1.2 SMS Detailed Data Flow Startup and Operation of SMS Receive Process and the SMS Template Caching

Startup of the SMS Receive Process (the process that reads all incoming messages packets and writes them to the shared memory segment) is launched by Subsystem Services. Initially, when the CCWS is booted, Subsystem Services launches the SMS Receive Process that creates the shared memory segment. Once the memory segment has been created , the SMS Receive Process starts receiving real-time message packets and records them to the SMS Receive Cache (shared memory segment). When a SMV viewer is started, the viewer's dedicated read thread starts retrieving those packets from the SMS Receive Cache for viewer display(this is accomplished through a SMS Retrieve API that retrieves new messages written to the memory segment). Next, the viewer determines if the message packet that was just received had previously been received and if so, the message template is obtained from the SMS Online Message Cache (shared memory area for received/retrieved message templates) and then used during translation for message display. If the SMS Online Message Cache does not contain a message format that was previously received, the message format is extracted from the CMS Online Catalog(the physical message index file). Reference the following figure:

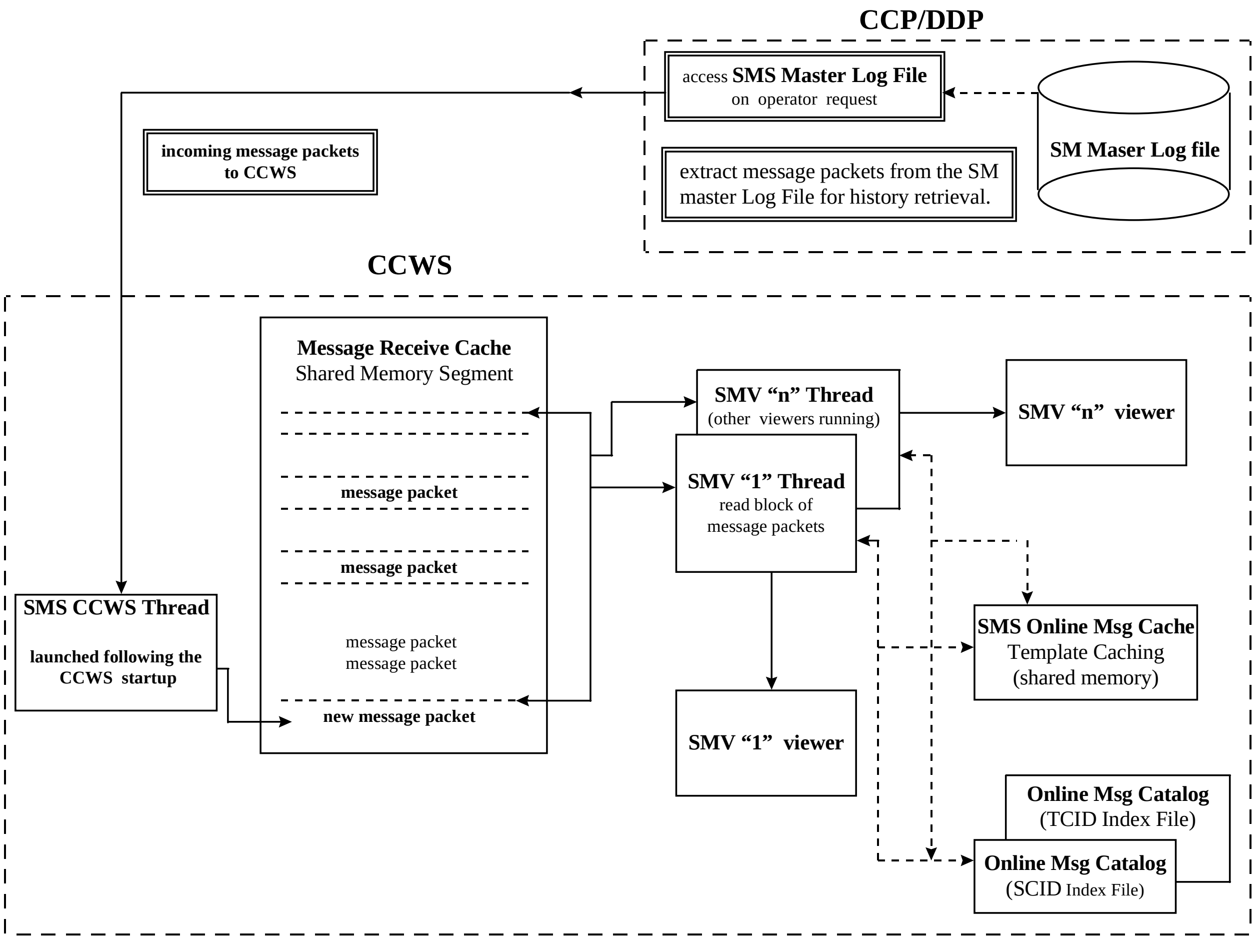


Figure 6. SMS Detailed Data Flow Startup and Operation of CCWS Receive Process and SMS Template Caching

As depicted in the previous diagram, there are two shared memory segments. The larger segment is known as the SMS CCWS Shared Memory Segment and is used as a repository for incoming message packets. As each message packet is receive by the CCWS, the SMS CCWS Receive Process reads those packets and writes them to the shared memory segment–the SMS CCWS Shared Memory Segment is also known as a FIFO message queue. As message packets are added to the CCWS Shared Memory Segment, the SMV viewer concurrently reads (through the use of a SMS retrieve API) and translates those packets for viewer display. During translation, the SMS Template Cache is used to determine the correct format for the displayable message and to reduce the latency time involved in accessing the physical Index file on disc. If the message template is not contained within the cache(the type of message packet has not been received prior to the viewer startup), the actual Message Index file must be used to translate the message packet. Prior to the translation of the packet, the message template used to translate the message packet is added to the SMS Template Cache for subsequent message translations. Additionally, if multiple viewers are active, all viewers will first task the SMS Template Cache segment prior to accessing the physical index file in order to reduce the number of hits to the physical disc.

1.3.2 SMS External Interfaces

1.3.2.1 SMS Message Formats

- Message number (#) does not exist within the message catalog: catalog name.
  (display the message packet)
- Unable to format message number (#).
  (list the reason for inability to display, and display the message packet)
- Message Catalog (catalog name) does not exist: display the path to the catalog and name

1.3.2.1 SMS Display Formats

N/A.

1.3.2.3 SMS Input Formats

N/A.

1.3.2.4 Recorded Data

The Message packet payload is recorded on the SDC. If a message request originated at the Gateway (via the SMS_SEND_SYSTEM_MESSGAE API), the packet gets forwarded to the CCWS(s). Both packets will get recorded at the SDC. If a message request originates on any other platform, only one packet will get recorded at the SDC.

Name of Recorded Data	Recording Type	SDC	Local
Message packet payload	C-C packet payload	X	

1.2.4.1   Service Printer Formats

N/A.

1.3.2.6 Interprocess Communications (C-to-C System Communications)

When an application issues the SMS_Send_System_Messgae API, SMS builds the following payload packet based on the passed parameters (i.e., message inserts) for the desired message.
* Note: The CC header packet fields (e.g., GMT2, Destination 2 fields) will be utilized
to retain these values (Original GMT, Source CPU_ID, Source RSYS, Source APP_ID when the original message was sent out (e.g., from the Gateway).

CC packet header	40 bytes
Message Number	2 bytes
Message Catalog	***1 byte (1 = SCID; 2 = TCID)***
CSC	***1 byte (applicable for "common msgs" only) (0 = N/A, non-zero = CSC number)***
Status Code	1 byte
# Inserts	1 byte
Insert 1 Type	1 byte (e.g., 0 = ASCIIZ_INSERT, 1 = INTEGER_INSERT, etc)
Insert 1	length dependent upon Insert Type
Insert 2 Type	1 byte
Insert 2	length dependent upon Insert Type
. . .	
Insert n Type	1 byte
Insert n	length dependent upon Insert Type

512 bytes max

Figure 1. Message packet.

The Message Packet body can be variable length since there can be from 0 to 30 inserts and the length is dependent upon insert type (e.g., ASCII, Decimal, Hex, etc.)

The table below shows the various Insert Types that can be declared in the Message Text field of the web Message Submission form.

CCMS System Msg insert type	CLCS insert declaration (utilizing sprintf type)	Comments
ASCII	%s	string is null-terminated
Decimal	%d	(16-bit request)
	%u	unsigned decimal
Floating Point	%f	[-]mmm.nnnnn notation
(new)	%e	[-]m.nnnnnne[+]xx notation
HEX	%x	uppercase HEX representation
OCTAL	%o	Octal representation
Binary	%b	Binary representation
CDT	%cdt	ddd:hhmm/ss notation
GMT	%gmt	hhmm/ss.sss notation
(new)	%mid	Message ID, uppercase HEX
Double-Word (64-bits) *(utilizing sprintf syntax where appropriate)*		
(new)	%lld	Decimal double-word
	%llu	"
(new)	%lle	Float double-word
	%llf	"
	%llg	"
(new)	%llx	uppercase Hex double-word
(new)	%llo	Octal double-word
(new)	%llb	Binary double-word

Figure 2. Message Insert Type Definitions

<u>Example:</u>
Below is an example:

1. **Message 197 as it would appear in the SDC Message Catalog database and the RTPS Message Index file:**

```
Gateway %s error signal from HIM during command issue.
        T/R Status Register = %x  HIM %o  CARD %o  FUNC %o
        HIM Status Register = %x.
```

2. **The SMS_Send_System_Messgae API**

```
sms_send_system_message (
        CGS_ERROR_SIGNAL_FROM_HIM, 6,
        ASCIIZ_INSERT, "GS1A",
        INTEGER_INSERT, 0x5A63,
        INTEGER_INSERT, 07,
        INTEGER_INSERT, 03,
        INTEGER_INSERT, 06,
        INTEGER_INSERT, 0x6B5F)
```

where 6 = number of inserts
"GSIA" is the ASCIIZ insert for the failed Gateway
"0x5A63" is the HEX number of the Transmitter/Receiver Status Register Contents
"07" is the OCTAL number of the HIM Number
"03" is the OCTAL number of the CARD Number

“06” is the OCTAL number of the FUNCTION CODE of HIM
“0x6B5F” is the HEX number of the HIM Status Register

3. The formatted message will appear as follows (reference SMV for prepennded information):

Gateway GS1A error signal from HIM during command issue.
T/R Status Register = 5A63 HIM 7 CARD 3 FUNC 6
HIM Status Register = 6B5F.

1.3.2.7 SMS External Interface Calls (e.g., API Calling Formats)

This is the data that is sent to SMS modules via a calling mechanism (e.g., API call)

- sms_format_message
  The sms_format_message API formats the message for display by the SMV viewer.
  Syntax: long sms_format_message(char *Rsys, int message_number, struct msgInsert *Inserts,
  SMS_PACKET_INFO_TYPE *out, struct MsgIndex *msgIndex, long rc)

- sms_build_message_packet
  The sms_build_message_packet API builds the C-C packet payload to be sent across the network.
  This is an internal API used by SMS as well as utilized by the Gateway. Status is returned in the packet body.
  Number of Bytes (actual size of packet_body) is returned from the function call.
  Syntax: int sms_build_message_packet (char * Rsys, int message_number,
  SYS_MSG_PACKET_PTR_TYPE sms_packet *, va_list argptr, int number_inserts)

- sms_get_message_help
  The sms_get_message_help API reads the HELP text associated with a message.
  Syntax: int sms_get_message_help (char *Rsys, char * message_buffer, int buffer_size, int message_number)

- sms_get_message_csc_details
  The sms_get_message_csc_details API obtains the details on the specific message for the given csc.
  Syntax: int sms_get_message_csc_details (char *Rsys, int message_number, char * message_buffer,
  int buffer_size)

- sms_get_message_packet_from_stream
  The sms_get_message_packet_from_stream reads a message packet from the stream socket and returns the number of bytes received.
  Syntax: int sms_get_message_packet_from_stream(SMS_MSG_PACKET_TYPE *packetBuffer,
  int sizeofpacket)

- sms_get_message_insert_from_database
  The sms_get_message_insert_from_database gets the message inserts and returns a success or failure status.
  Syntax: int sms_get_message_insert_from_database(cahr *Rsys, int msgNumber,
  INSERT_RECORD_PTR_TYPE * insertBuffer)

- sms_send_system_message
  The sms_send_system_message API issues a TCID message request to be sent across the network.
  SMS builds a packet payload (calling sms_build_packet), then sends the C-C type packet across the network.
  Syntax: int sms_send_system_message (char *Rsys, int message_number, int number_inserts,
  int insert_1_type, insert_1, . . .)

Message Index File

Please Reference ***Figure 3*** for a pictorial representation of the Message Index File.

The SCID Message Index File below is created as part of the System Build process, and the TCID Message Index File is created as part of the Test Build process. The SCID Message Index File is created from the SCID Message Catalog. The TCID Message Index File is created from the TCID Message Catalog. Both Message Catalogs contain all of the message text, help text and associated fields; e.g., severity level, alarm, CSCI. Both Message Index Files are indexed by Message Number. The (SMS) API must make 2 references to the Indexed File; the first reference is to the beginning part of the indexed file to retrieve the file byte offset to the location of the Message Text. The 2nd reference is to the actual file location of the Message Text to be retrieved.

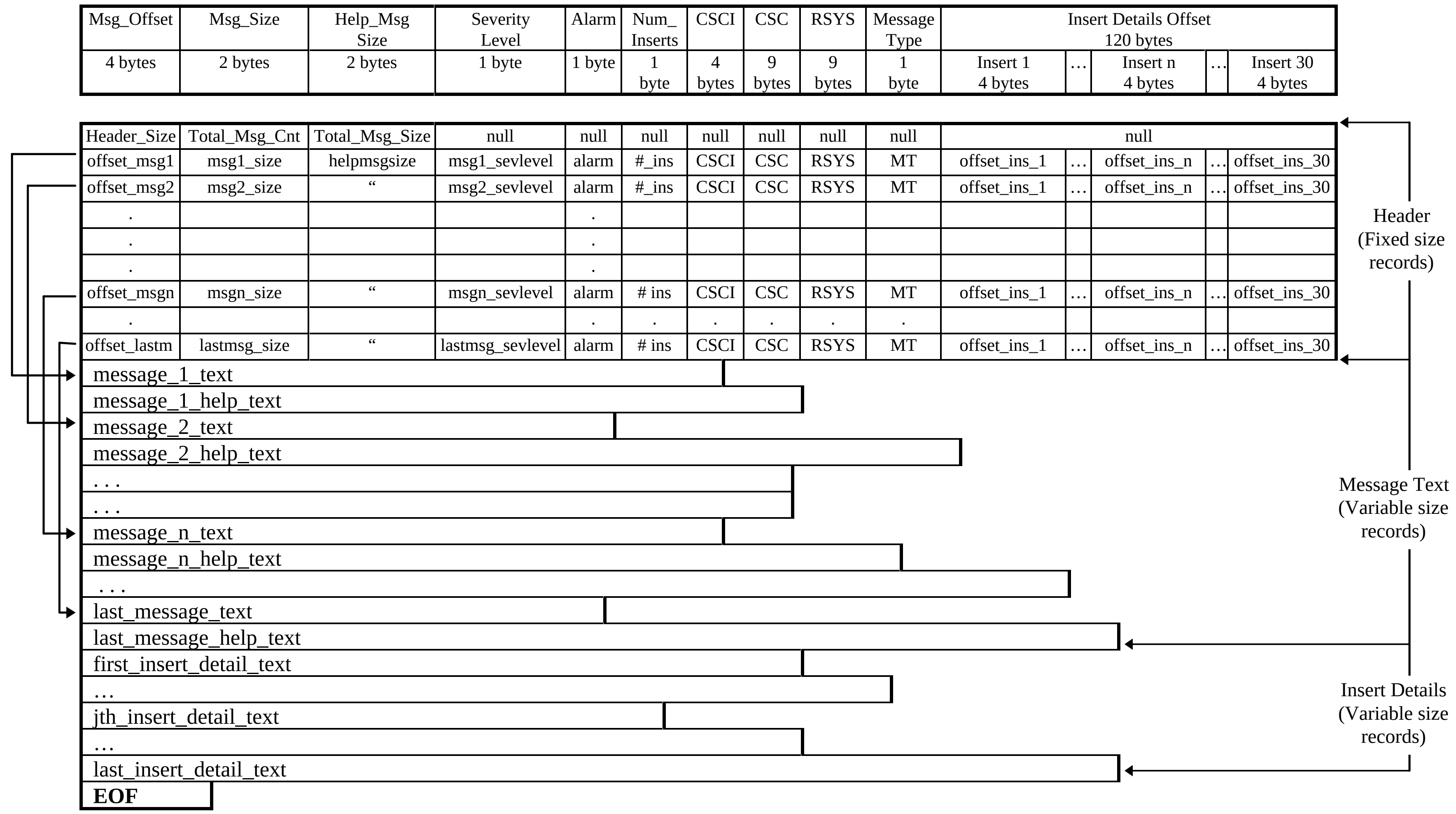
The Help text is retrieved as follows: the offset to the Help Text is calculated based on the Message Text offset + Message Text size. This gives the file byte offset to the Help Text.

Insert Details are appended to the end of the Message Index File. The Insert Details Offset section consists of an array of 30 positions. Each position contains the relative offset the Insert Detail is from the end of the last Message Help Text entry. Therefore, actual Insert Detail positions are calculated based on the Header Size + Total Msg Size + Insert Offset. This scheme allows details to be reused by multiple messages within the same Message Index File. The Total Msg Size value will be retrieved the first time it is needed, and will be stored in memory for subsequent uses by SMS. This keeps the number of disk accesses to the Message Catalog at two, after the initial retrieval.

The CSCI, CSC, and RSYS are null-terminated strings. The actual text is 3, 8, 8 characters long respectively.

The Message Type is a 1 byte integer, and corresponds to one of the following constants:
1 = "Summary"
2 = "Intermediate"
3 = "Details"
4 = "other"

Total_Msg_Cnt represents the total allowed number of messages in the Catalog. This is done for reuse of deleted message numbers. Therefore, a new message number may occupy the same space held by a previously deleted message



Msg_Offset	Msg_Size	Help_Msg Size	Severity Level	Alarm	Num_ Inserts	CSCI	CSC	RSYS	Message Type	Insert Details Offset 120 bytes				
4 bytes	2 bytes	2 bytes	1 byte	1 byte	1 byte	4 bytes	9 bytes	9 bytes	1 byte	Insert 1 4 bytes	…	Insert n 4 bytes	…	Insert 30 4 bytes

Header_Size	Total_Msg_Cnt	Total_Msg_Size	null	null	null	null	null	null	null	null				
offset_msg1	msg1_size	helpmsgsize	msg1_sevlevel	alarm	#_ins	CSCI	CSC	RSYS	MT	offset_ins_1	…	offset_ins_n	…	offset_ins_30
offset_msg2	msg2_size	“	msg2_sevlevel	alarm	#_ins	CSCI	CSC	RSYS	MT	offset_ins_1	…	offset_ins_n	…	offset_ins_30
.				.										
.				.										
.				.										
offset_msgn	msgn_size	“	msgn_sevlevel	alarm	# ins	CSCI	CSC	RSYS	MT	offset_ins_1	…	offset_ins_n	…	offset_ins_30
.				.	.	.	.	.	.					
offset_lastm	lastmsg_size	“	lastmsg_sevlevel	alarm	# ins	CSCI	CSC	RSYS	MT	offset_ins_1	…	offset_ins_n	…	offset_ins_30

Figure 3. Pictorial representation of the Message Index File, “*smcs_xxxx_messages.idx*”.

Note: The CSCI Insert Types are not required in the Index File.
The file byte offset to the Help Text is not required. Help Text offset is calculated as follows:
Help Text Offset = Message Text Offset + Message Text Size.
Insert Details offsets are relative to the end of the message and help text. Actual positions are calculated as follows:
Insert Details Position = Header Size + Total Message Size + Insert Offset.

1.3.3 SMS Test Plan

SMS system-level tests may be run in either or both the IDE or SDE environments. These tests are run on the basic CCWS, CCP, DDP and Gateway platforms. There are no special hardware configurations required.

SMS testing also requires a CLCS application or a CLCS like-application test tool that exercises the various SMS APIs.

The specific test cases that will be run include:

1. Send Message API from the Gateway to the CCP
2. Send Message API from the CCP to the CCWS
3. Receive Incoming Message API (from CCWS to System Message Viewer)
4. Get Message API (retrieve Message Text from Indexed File formatted with parameter inserts)
5. Get Message HELP API (retrieve Message Help Text from Indexed File)
6. Get Message SCS Details API (retrieve Message Details based on SCS from the message index file)
7. Send TCID Message API (send TCID message to the CCWS from the gateway.CCP/DDP)
8. Send Common Message API (send TCID message to the CCWS from the gateway.CCP/DDP)
9. Send mass influx of messages to the CCWS for viewer retrieval.
10. Send mass influx of the same messages to the CCWS for viewer retrieval.
11. Retrieve message concurrently while influx to SMS Receive Process.
12. Force error condition 1 (message number (#) dose not exist within the message catalog: catalog-name.
13. Force error condition 2 (Unable to format the message number (#) for display).
14. Force error condition 3 (Message catalog does not exist).

1.4 Appendix of Low Level System Architecture

- Message Receive Cache (message packet shared memory logic)
- Message Template Cache (template share memory logic)